

Prediction of Kitchen Preparation Time in Food Delivery

Thesis submitted by

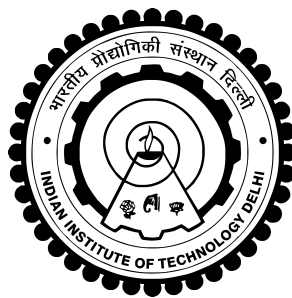
Manoj Kumar
2018CS50411

under the guidance of

Prof. Sayan Ranu & Prof. Amitabha Bagchi

*in partial fulfilment of the requirements
for the award of the degree of*

Bachelor and Master of Technology



Department Of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

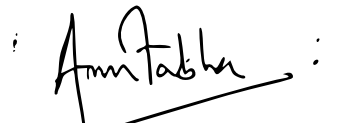
June 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **Prediction of Kitchen Preparation Time in Food Delivery** SUBMITTED TO **IIT-D**, submitted by **Manoj Kumar**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor and Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.



Prof. Sayan Ranu
Professor
Dept. of Computer Science and
Engineering
IIT-Delhi, 600 036



Prof. Amitabha Bagchi
Professor
Dept. of Computer Science and
Engineering
IIT-Delhi, 600 036

Place: New Delhi

Date: 24th June 2023

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my advisors, Prof. Sayan Ranu and Prof. Amitabha Bagchi, for granting me the invaluable opportunity to work on this wonderful project in the field of food delivery. Without their constant encouragement and guidance, this work would not have been possible.

Furthermore, I am sincerely thankful to Dominos India (Jubilant Foodworks Ltd.) for generously providing us with their dataset for conducting experiments and for sharing insightful inputs regarding the food delivery business.

I would also like to extend my appreciation to PhD scholar Anjali Gupta for her continuous and insightful inputs and advice throughout this project. Additionally, I would like to express my gratitude to my collaborator, [Umesh Parmar](#) for his valuable contributions to the project.

Lastly, I would like to express my gratitude to the Department of Computer Science and Engineering at the Indian Institute of Technology Delhi for granting us access to their cloud infrastructure, which were instrumental in carrying out this project.

Contents

ACKNOWLEDGEMENTS	i
LIST OF TABLES	iv
LIST OF FIGURES	vi
ABBREVIATIONS	vi
ABSTRACT	vii
1 Prediction of Kitchen Preparation Time	1
1.1 Introduction	1
1.2 Problem Statement	2
1.3 Dataset for the research	3
1.4 Why Machine Learning	3
1.5 Regression Models	4
1.6 Feature Creation	5
1.6.1 Model Output Value	5
1.6.2 Input Feature Creation	6
1.6.3 Final Model Input Features	7
1.7 Model Selection and Design	10
1.7.1 Final Model Results on Different Cities	12
1.7.2 Baseline Approach	13
1.7.3 Statistical Significance of the Comparison	21
1.7.4 Model Deterioration Window	22
2 Service Implementation for JFL	24
2.1 Introduction	24
2.2 Current Architecture of JFL & Requirements	24

2.3	Requirements of Dominos Team	25
2.4	New Architecture Design: Microservices or Monolithic?	25
2.5	System Architecture Design	27
2.5.1	KPT Model	27
2.5.2	KPTHandler	29
2.5.3	Get Prediction (Interface)	31
2.6	Docker for service Deployments	32
A	XGBoost Model for Regression	33
B	Springboot Framework for Java	35

List of Tables

1.1	Error comparison to check the feature importance. Random Forest regression model is used here. The train and test data split here is random among July 2022 data of Bhopal	7
1.2	Error comparison to check the feature importance. Random Forest regression model is used here. The train and test data split here is random among July 2022 data of Jaipur	8
1.3	Overall Performance on Test Data (22-31 July 2022) of Bhopal	14
1.4	Overall Performance on Test Data (22-31 July 2022) of Delhi	14
1.5	Error values (in seconds) on 28 July 2022 Bhopal data for different training periods for model deterioration check	23
1.6	Error values (in seconds) on 28 July 2022 Bhopal data for different training periods for model deterioration check	23

List of Figures

1.1 Feature Importance analysis on the Bhopal city over the data of July 2022	8
1.2 Feature Importance analysis on the Bhopal city over the data of July 2022	11
1.3 MAE Comparison Saywise	15
1.4 Difference of XGBoost and Baseline approach errors on (hour x store) pairs	15
1.5 Difference of XGBoost and Baseline approach errors on (day x store) pairs	16
1.6 Difference of XGBoost and Baseline approach errors on (day x store) pairs	17
1.7 Difference of XGBoost and Baseline approach errors on (day x store) pairs	17
1.8 5-second accuracy comparison between the XGBoost model and baseline approach	18
1.9 10-second accuracy comparison between the XGBoost model and baseline approach	18
1.10 20-second accuracy comparison between the XGBoost model and baseline approach	19
1.11 30-second accuracy comparison between the XGBoost model and baseline approach	19
1.12 60-second accuracy comparison between the XGBoost model and baseline approach	20
1.13 80-second accuracy comparison between the XGBoost model and baseline approach	20
1.14 t-statistic and p-value on the predicted time of the XGBoost model and the baseline on the testing data	21
1.15 t-statistic and p-value on the predicted time of the XGBoost model and the baseline on testing data during Normal Time and Rush Hours	22
1.16 t-statistic and p-value for the alternate hypothesis on whole testing data .	22
1.17 t-statistic and p-value for the alternate hypothesis during normal times . .	22
1.18 t-statistic and p-value for the alternate hypothesis during rush times	23
2.1 Architecture Design of Microservices	27
2.2 Architecture Design of KPT Handler Microservice	29
2.3 Event Publisher hitting KPT Handler	31

ABBREVIATIONS

IITD	Indian Institute of Technology, Delhi
KPT	Kitchen Preparation Time
JFL	Jubilant Foodworks Ltd.
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
XGBoost	Extreme Gradient Boosting

ABSTRACT

KEYWORDS: Machine Learning ; XGBoost ; Regression Model ; Food Delivery ; Kitchen Predictions ; Springboot Server ; Docker ; APIs.

Today, online food orders and delivery market is valued at above 83 billion euros [7]. Several companies such as Uber Eats, Swiggy, Zomato etc. are available in the market where users can place an order from any restaurant, and the delivery partner associated with the company delivers the order. Apart from these apps, restaurant franchises and chains such as Domino's, KFC, pizza hut etc. have multiple outlets around the world. A franchise like Domino's takes orders on their app and delivers with their own delivery partners. They do not use third-party delivery riders.

In Domino's, there are multiple outlets across the big cities, and some delivery partners are associated with each Domino's outlet, who deliver only those orders which come to their associated outlet.

We aim to un-bundle these delivery riders to their stores in such a way that riders do not constraint themselves to a particular restaurant only, but they can also deliver the nearby other restaurants of the same company as well. We aim that each new order assigns to a store and a delivery rider in such a way that the overall delivery time reduces for all the orders.

While assigning the store and a proper rider assignment to each order, it is important to predict how much time the order is going to take in order to prepare the food, so that we can do a better assignment. Predicting the food preparation time is a sub-problem of the store and the rider assignment problem.

Chapter 1

Prediction of Kitchen Preparation Time

1.1 Introduction

^[1] The restaurant industry has experienced a rapid increase in demand and competition in recent years, necessitating the implementation of innovative solutions to improve operational efficiency. The amount of time it takes to prepare and serve customers' orders is a crucial factor influencing their satisfaction. Accurate estimation of food preparation time can facilitate kitchen operations, optimize staff allocation, and minimize customer wait times, resulting in increased customer satisfaction and revenue.

This thesis presents a solution that utilizes the power of machine learning to more precisely determine food preparation time. By analyzing historical data comprising a variety of factors, the machine learning model can discover intricate patterns and correlations in order to generate precise time estimates.

In Domino's like stores, when a customer places an order online through their app, the food delivery process unfolds in four discrete steps:

1. Assign the order to the nearest outlet.
2. A delivery vehicle is assigned. (assignment time).
3. The delivery vehicle drives to the restaurant to pick up the next order. (first mile time)
4. Preparation and packaging of the food by the restaurant. (preparation time)
5. Dropping off the order at the customer's location. (last-mile time)

If a delivery vehicle is already available at the outlet, the first-mile time is zero. But if there is no delivery vehicle available at the outlet (i.e. all vehicles are delivering some other orders), then it waits for a delivery vehicle to arrive. This waiting time is first-mile time. Here

$$\text{Delivery Time} = \max \{ \text{assignment time} + \text{first-mile time}, \\ \text{preparation time} \} + \text{last-mile time}.$$

¹Umesh Parmar ^[10] worked broadly on the raw dataset provided by Jubilant Foodworks Ltd. ^[13] and prepared the useful dataset after a deep analysis on the raw data which is used in my thesis work ^[1.3]. Umesh also contributed in creating and training the machine learning Model to predict the kitchen preparation time.

Optimizing (minimizing) the overall delivery time is the key to ensuring a positive customer experience. Most of the time, the first-mile time is lesser than food preparation time. Hence, food preparation time overcomes the first-mile time. Hence, to predict the Delivery Time more precisely, we need to predict the kitchen preparation time (KPT) in advance.

1.2 Problem Statement

In the above delivery time equation, the first mile and the food preparation happen parallel; for a better assignment of store and delivery rider, we need to have a better idea of how much time the order will take to be baked in each nearby store.

Existing manual methods of estimating preparation time, such as relying on the experience of kitchen staff or using generic guidelines, are subjective and prone to error. They fail to consider various factors that can significantly impact the time required to prepare an item, such as its complexity, ingredient quantities, cooking techniques, and the workload of the kitchen.

To address this problem, the objective of this thesis is to develop a machine learning model that can accurately predict the kitchen preparation time of each item in a store. By leveraging historical data that encompasses a wide range of relevant factors, the model will learn complex patterns and relationships to provide precise estimates.

The successful development and implementation of such a model would offer several benefits. Firstly, it would enable store owners and managers to optimize kitchen operations by effectively allocating resources and planning workflows. This would lead to reduced waiting times, enhanced customer satisfaction, and increased revenue. Secondly, it would provide valuable insights into the factors that influence preparation time, enabling informed decision-making for menu planning, inventory management, and staff scheduling.

However, building an accurate and reliable prediction model for kitchen preparation time poses several challenges. The model needs to account for the diverse range of food items typically prepared in a store, each with its unique characteristics and complexities. It must also handle real-time data updates to adapt to changing ingredients, recipes, and cooking processes. Additionally, scalability and performance are crucial considerations to ensure the model can handle the volume of requests and deliver predictions in a timely manner.

The aim of this project, therefore, is to overcome these challenges by developing a robust machine learning model capable of accurately predicting the kitchen preparation time of items in a store. By doing so, this project will contribute to improving operational efficiency,

customer satisfaction, and profitability in the food retail industry.

1.3 Dataset for the research

²We got the data for the following cities and time period from Dominos India (Jubilant Food Works Limited [13]):

1. Bhopal (17 Dec 2021 - 31 Dec 2021 & 1 July 2022 - 31 July 2022)
2. Delhi (17 Dec 2021 - 31 Dec 2021 & 1 July 2022 - 31 July 2022)
3. Jaipur (17 Dec 2021 - 31 Dec 2021 & 1 July 2022 - 31 July 2022)
4. Kolkata (17 Dec 2021 - 31 Dec 2021 & 1 July 2022 - 31 July 2022)
5. Mumbai (17 Dec 2021 - 31 Dec 2021 & 1 July 2022 - 31 July 2022)

The data contains the information of each order and each item with the timestamp of the order placed, order baked etc.

Using these timestamps, the preparation time is defined as follows:

$$\text{Preparation Time} = \text{OVEN_IN_TIME} - \text{KITCHEN_DISPLAY_TIME} + 7\text{min (oven time)}.$$

Here Kitchen Display Time is the time when the kitchen is informed about the order item. Oven In Time is the time at which the item is put inside the oven. The item remains in the oven till 7 minutes. The Oven Time is static for all kinds of baking items, Since non-baking items do not take any time to bake, we exclude such items in training our model.

Hence, our model should predict only the variable part of the kitchen preparation time: Oven In Time - Kitchen Display Time. To Calculate the overall Kitchen Preparation time, we simply add 7 minutes to the predicted output variable.

1.4 Why Machine Learning

Since Dominos, like companies, bake similar kinds of items and the kitchen preparation time of an item largely depends on the current load in a store at a time, or in other words, it might depend on depends on the historical data of items baked just before the current item.

²The Dataset for the research is prepared by Umesh Parmar, please find the detailed analysis of the data in the thesis.[10]

The kitchen in a store can be seen as a queuing system, where the orders arrive, wait in a queue, and are processed by kitchen staff before being served to the customers.

By analyzing historical data and observing the queuing behavior in the kitchen, it is possible to build a queuing model that can provide predictions for kitchen preparation time. There are several reasons why Machine Learning models are better than a queuing model for the prediction of kitchen preparation time in our case:

1. It is important to note that building an accurate queuing model for kitchen preparation time may require lots of data of different types such as staff performance, ovens in a restaurant, and other dynamic elements that can influence the accuracy of the predictions. Dominos does not have that much information in one place for all its stores in India. Besides, A model using machine learning can dynamically train itself based on all the factors that impact preparation time.
2. Incorporating Non-linear Relationships: Machine Learning models can capture non-linear relationships between input variables and the kitchen preparation time. They can learn complex patterns from the data and make predictions accordingly. In contrast, queuing models often rely on assumptions of linearity or certain distributional properties, which may limit their ability to capture the full complexity of the kitchen preparation process.
3. Adaptability and Learning: Machine learning models can adapt and learn from new data, continuously improving their prediction accuracy over time. They can adjust their predictions based on changing circumstances, seasonal variations, or evolving kitchen dynamics. Queuing models are typically static and may require manual adjustments to accommodate changes, making them less adaptable in dynamic environments.
4. Robustness: Machine learning models can handle uncertain or incomplete data and make reasonable predictions or decisions based on the available information. They can also identify anomalies or unusual patterns that may indicate potential issues. Queueing systems often rely on deterministic rules, which can be less robust in handling unexpected situations or outliers.

Also, machine learning models are capable of computing the distributions of queueing systems with great accuracy [5] (c.f. It is cited from a journal that appeared in SIGMETRICS, a top-rated international conference). On the other side, there is research showing that the queuing analysis can do better only in simple scenarios (c.f. [6]). The complex interplay of different factors that determines preparation time in a Dominos kitchen is better approached via machine learning methods.

1.5 Regression Models

For a given set of input features, we aim to predict the kitchen preparation time of an item. Regression is a statistical modeling technique used to predict continuous numerical values

based on input variables. Here the kitchen preparation time is a continuous numerical value, the prediction output. The regression model can take into account the quantitative nature of the input variables and capture the patterns and trends in the data to estimate the expected kitchen preparation time. It allows for predicting a continuous numerical value rather than a discrete outcome.

We researched on the following three regression models for predicting Kitchen preparation time for dominos Data:

1. Neural Network
2. Random Forest
3. Gradient Boosting

Gradient Boosting model gives the best results for JFL (Dominos) Dataset, hence we will only see the XGBoost model only.

1.6 Feature Creation

Using the raw dataset given to us by Dominos Team, we created features which can be used to feed to the machine learning model to predict the output variable (i.e. Kitchen preparation time). The following things should keep in mind while creating the input features:

1. Features should cover all aspects of the process which can affect the kitchen preparation time.
2. Relevance to the problem: Features should be selected based on their relevance to the kitchen preparation time. They should capture some meaningful information.
3. Dimensionality Reduction: To mitigate the curse of dimensionality and improve the model performance, we need to reduce the number of features while preserving relevant information if the feature space is high-dimensional.
4. Data Availability: All features should be available in both training and test dataset. If they are not, we should be capable of creating the features in real-time.

1.6.1 Model Output Value

We aim to predict the kitchen preparation time of each item using our model. To train our model, let's first create the output variable of the model (i.e. Kitchen Preparation time of an item)

For non-baking items (beverages, dippings etc.), we take preparation time = 0 hence we exclude these items while predicting the preparation time.

The preparation time of the order is calculated as follows:

Let $O = \{o_1, o_2, \dots, o_n\}$ be the order with n baking items.

$$\text{prepTime}(o_i) = (\text{ovenTime}(o_i) - \text{kitchenDisplayTime}(o_i)) + 7 \text{ minutes}$$

Here:

Kitchen Display time: The time when the order is displayed in the kitchen.

Oven Time: The time when the baking item was put into the oven.

7 minutes is the standard time for a baking item to be in the oven. Since, this is a constant time, we do not include it in the model output variable of the machine learning model.

$$\text{ModelOutput}(o_i) = \text{ovenTime}(o_i) - \text{kitchenDisplayTime}(o_i)$$

or,

$$\text{prepTime}(o_i) = \text{ModelOutput}(o_i) + 7 \text{ minutes}$$

Overall preparation Time of the order:

$$\text{prepTime}(O) = \max \{ \text{prepTime}(o_1), \text{prepTime}(o_2), \dots, \text{prepTime}(o_n) \}$$

or,

$$\text{prepTime}(O) = \max \{ \text{ModelOutput}(o_1), \text{ModelOutput}(o_2), \dots, \text{ModelOutput}(o_n) \} + 7 \text{ minutes}$$

Our machine learning model predicts the value of **ModelOutput**(o_i) and then we can compute the preparation time of the whole order.

1.6.2 Input Feature Creation

We can extract the following information/features from the provided dataset for each item:

1. **f1:** Store id
2. **f2:** Item Type (Menu Code)
3. **f3:** Timeslot of Kitchen Display Time
4. **f4:** Count of all baking items of all orders received by the store in the last 30 minutes.
5. **f5:** Avg prep time of the same item in the same store in last 60 minutes.
6. **f6:** Length of the queue in respective store

Note:

Feature 1,2 and 3 are categorical data and need to be converted into numerical data for some ML models (For example NN etc). To save the ML model from misinterpreting that there is some hierarchy in them, we use Features 1, 2, and 3 as one-hot encoded.

We compare these features to check which features are important and which are not. To do this, we apply Random Forest regression model and check the RMSE error with a random split of 80% training data and 20% testing data.

Table 1.1: Error comparison to check the feature importance. Random Forest regression model is used here. The train and test data split here is random among July 2022 data of **Bhopal**

	RMSE on Train Set	RMSE on Test Set
No Removal	15.514185	40.665234
Remove f1 (store id)	16.560230	43.221314
Remove f2 (Item Type)	15.315657	39.887082
Remove f3 (Timeslot of kitchen display time)	16.181006	42.324128
Remove f4 (count past orders)	16.507189	43.791777
Remove f5 (previous prep time)	20.318836	49.140916
Remove f6 (queue length)	19.118799	49.808505
Remove (F2 F5)	32.568478	43.777195
Remove (F1 F4)	18.530093	48.102677
Remove (F5 F6)	43.720981	89.521577
Remove (F4 F6)	24.519576	63.931472
Remove (F4 F5)	40.338082	63.544194
Remove (F4, F5, F6)	264.163058	279.146601

In Table [1.1](#), figure [1.1](#) and Table [1.2](#), we can observe that **Item type** feature (f2) is not affecting the accuracy of the model. It means that the item preparation time in Dominos restaurant does not depend on what type of item they are preparing. It is because, most of the items are pre-prepared in Dominos store, they just need to add some toppings to their pizzas and put it into the oven.

Hence, we can exclude the item type feature from the model inputs.

1.6.3 Final Model Input Features

Feature 1: Store Id

Numerical Categorical Feature. Mapped each store id to a numerical value. This is because the Kitchen preparation time depends on the number of staff in the store, no. of ovens in

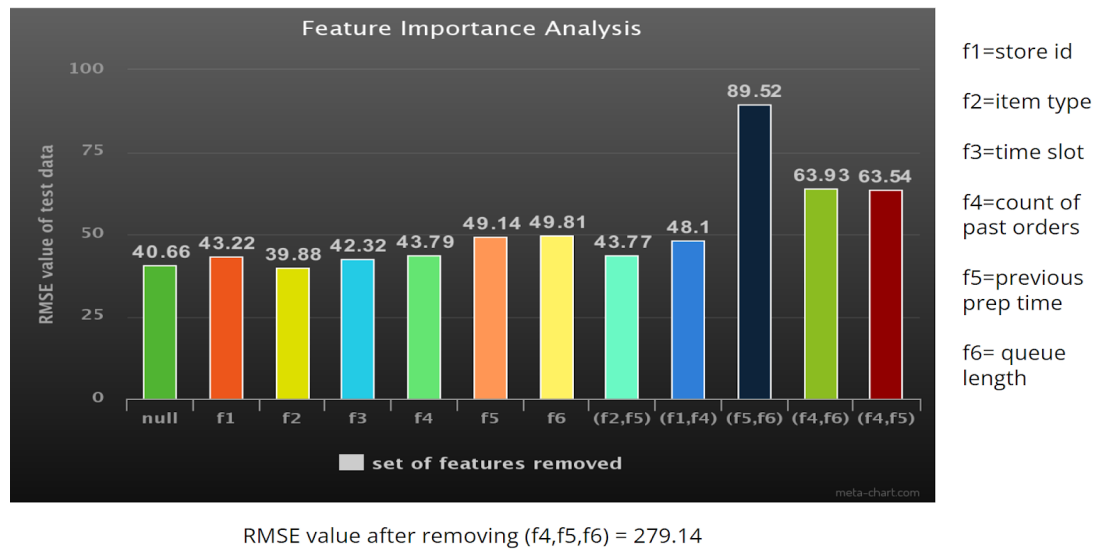


Figure 1.1: Feature Importance analysis on the Bhopal city over the data of July 2022

Table 1.2: Error comparison to check the feature importance. Random Forest regression model is used here. The train and test data split here is random among July 2022 data of **Jaipur**

	RMSE on Train Set	RMSE on Test Set
No Removal	37.08	75.14
Remove f1 (store id)	33.32	75.14
Remove f2 (Item Type)	35.12	74.93
Remove f3 (Timeslot of kitchen display time)	40.84	80.33
Remove f4 (count past orders)	32.21	76.45
Remove f5 (previous prep time)	70.35	173.23
Remove f6 (queue length)	38.84	85.04
Remove (F2 F5)	82.15	119.48
Remove (F1 F4)	37.04	80.85
Remove (F5 F6)	141.26	219.90
Remove (F4 F6)	41.29	92.96
Remove (F4 F5)	122.21	215.83
Remove (F4, F5, F6)	214.49	237.643

the store, etc. which are specific to a store. Since we do not have data about the staff and ovens of a store, we take the store id as a feature to neutralize the effect of the store-based elements. Machine learning models do not support string values, hence we map each store id to a numerical value from 1 to the total no. of the store.

We convert it to one-hot encoded values so that the model does not misinterpret it that there is some hierarchy in them.

Feature 2: Day of the week

Range [1, 7] categorical feature. Dominos Data has a different load pattern on each day of the week. Generally, in tier-1 cities, where people have holidays on Saturday and Sunday, we see a high number of orders on weekends as compared to weekdays.

Hence, to let the model understand the effect of day of the week, we add it to the features. Again, we convert it to one-hot encoded values so that the model does not misinterpret it that there is some hierarchy in its values.

Feature 3: Timeslot (Hour) of Kitchen Display Time (Order Time)

Range [1, 24] categorical feature. Dominos Data has a different load pattern on the morning, lunchtime, or evening hours. Hence, the load on the store might increase at a specific time every day as many people might order pizza for their dinner the load might increase during evening hours. The model should be trained to understand this pattern. Hence we add it as a feature.

Again, we convert it to one-hot encoded values so that the model does not misinterpret that there is some hierarchy in the hours of the time.

Feature 4: Count past Items

Count of all baking items of all orders received in respective store in the last 30 minutes. It signifies the load of the store at a time.

Feature 5: Lift in past orders

Ratio of number of items received in last 30 minutes window and number of items received in last 60-30 minute window. For example, if a store gets total 100 items in 1:00PM - 1:30PM and 200 items in 1:30PM - 2:00PM, then the lift at 2:00PM will be $200/100 = 2$. If the restaurant has just opened, take it 1.

Feature 6: Average KPT

Average KPT (Kitchen Preparation Time) in the same store in the last 1 hour.

Feature 7: Average KPT of last 7 days

Average KPT (Kitchen Preparation Time) in the same store in the last days days.

Feature 8: Average Timeslot Daywise KPT of last 30 days

Average KPT in the same store in the same timeslot in last 30 days (1 month) on the same day of the week. (For example, if someone ordered an item at 1:30 PM on Monday, then we'll take an average of all past items that were placed between 1:00 PM to 2:00 PM on previous Mondays of the month)

Feature 9: Length of the Queue

Total number of pending orders in the restaurant which are not baked yet.

Figure [1.2](#) shows the final Feature Importance graph. It shows that the most important feature among all features is Feature 6 (Average Kitchen Preparation Time in the same store in last 1 hour).

1.7 Model Selection and Design

We use the XGBoost Regression model to make the predictions. XGBoost (Extreme Gradient Boosting) [\[4\]](#) is an ensemble method that combines predictions of multiple decision tree models to create a more robust overall prediction. It uses a boosting technique, where each subsequent model corrects the mistakes of the previous models. [\[2\]](#)

It first builds a single decision tree, often called the base or initial model. The subsequent trees are built to minimize the errors or residuals of the previous trees.

XGBoost uses a specific objective function for regression tasks, which measures the difference between the predicted and actual values. The objective function is designed to be differentiable, allowing gradient-based optimization.

Each new decision tree is trained to predict the residuals (the difference between the actual and predicted values) of the previous trees. The gradients of the objective function with respect to the predictions are computed to determine the direction and magnitude of the updates for each tree.

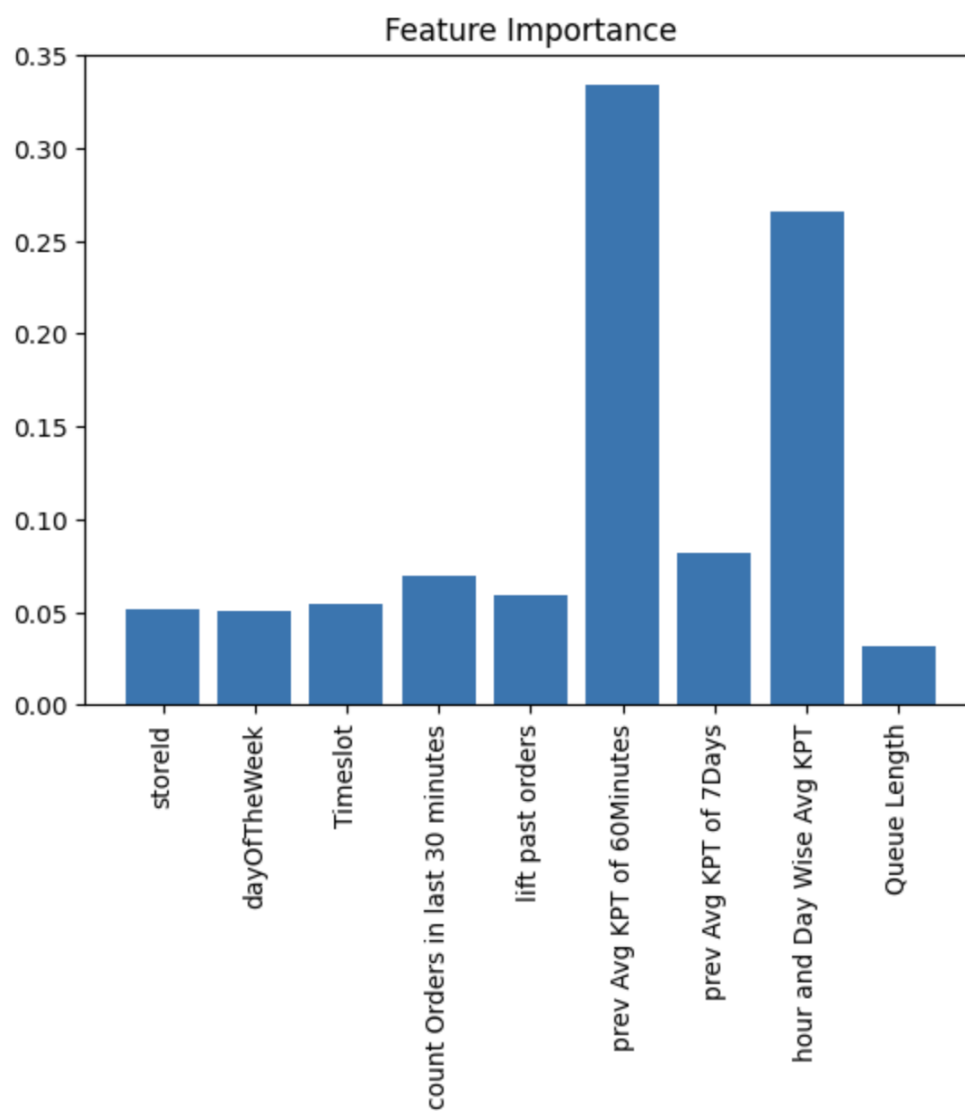


Figure 1.2: Feature Importance analysis on the Bhopal city over the data of July 2022

1.7.1 Final Model Results on Different Cities

Due to the business constraints of JFL Team (Dominos India), we have to exclude item type as a feature (because the Dominos Business team wants to predict the preparation time before preparing the cart).

We applied a grid search to find the best hyperparameters for the XGBoost model [9]. The hyperparameters are as follows :

1. `colsample_bytree`: 0.7 (This is the sub-sampling ratio of features when constructing each tree in the gradient boosting process, i.e., 70% of the features will be randomly chosen for each tree.)
2. `learning_rate`: 0.1 (It scales the contribution of each tree in the ensemble. Learning rate is multiplied by the predicted values of the previous trees and added to the current model's prediction.)
3. `n_estimators`: 100 (Number of boosting iterations or total number of trees in the ensemble.)
4. `subsample`: 0.7 (70% samples will be selected randomly for each tree. It helps overfitting and improve generalization performance.)
5. `tree_method`: 'hist' (It constructs histograms of the features and uses these histograms to find the best split points. This method also supports various optimizations, such as weighted quantile sketch and GPU acceleration, which makes it faster for large datasets)
6. `loss function`: regression mean squared error.

With these parameters and the training data period of 1 - 21 July 2022 of Bhopal, the model is trained and we got the following results:

```
Test score: 0.3729048807849119
ACCURACY ON Train DATA :
RMSE score: 67.79383904037947
ACCURACY ON Test DATA :
RMSE score: 90.14257903673608
```

The results are for Bhopal city:

- Training data: 1st July to 21st July 2022
- Test Data: 22nd July to 31st July 2022

Individual models are trained for each city.

1.7.2 Baseline Approach

Referring to the feature importance graph in figure [1.2](#), we can see that the predictions heavily depend on an item's previous kitchen preparation time in the last 1 hour in the respective store. We assume it is the baseline approach.

Since the baseline approach is the average of the last 1 hour and the kitchen preparation time of an item is more or less the same for most of the items in Dominos, like chain restaurants since they follow a fixed process to bake an item. Hence, the baseline approach also gives very good results in regular load hours.

When the load on the stores suddenly increases in peak hours, that time baseline approach can not predict accurately, but the Machine Learning model can predict during such times with more accuracy.

Model Performance comparison

We train the XGBoost model in Bhopal city for the following time:

- Training Period: 1 - 21 July 2022
- Testing Data 22 - 31 July 2022

The ML Model and Baseline approaches are compared on the following parameters:

1. Mean Absolute Error: This is the mean of the absolute values of all errors (difference between actual and predicted value).

$$MAE = \frac{\sum_{i=1}^n |actual_i - predicted_i|}{n}$$

2. Root Mean Square Error: RMSE of the model is computed as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (actual_i - predicted_i)^2}{n}}$$

3. Median Absolute Error: This is the median of errors of all items.
4. Std Deviation in Absolute Error: This is the standard deviation from the mean of errors of all items.

$$STD = \sqrt{\frac{\sum_{i=1}^n (e_i - MAE)^2}{n}}$$

where $e_i = |actual_i - predicted_i|$

The parameter values for Bhopal are given in table [1.3](#) and for Delhi are shown in table [1.4](#)

Table 1.3: Overall Performance on Test Data (22-31 July 2022) of **Bhopal**

	Baseline Approach error (in seconds)	XGBoost Model error (in seconds)
MAE	62	61
RMSE	94	90
Median Error	39	40
Std Dev	72	66

Table 1.4: Overall Performance on Test Data (22-31 July 2022) of **Delhi**

	Baseline Approach error (in seconds)	XGBoost Model error (in seconds)
MAE	94	89
RMSE	129	122
Median Error	69	68
Std Dev	94	83

Now, if the model and baseline approaches are compared on each day of test data (22 - 31 July) on the basis of mean absolute error, we get the figure [1.3](#). It shows that error is high on weekends, which is because of high rush on weekends. Also, the machine learning model is performing far better than the baseline approach during high rush. We can also check the difference between the mean absolute errors of xgboost model and baseline approach. (i.e. difference = MAE of the Baseline Approach - MAE of the model) If the difference is positive, that means Error in baseline is greater than the error in the XGboost model.

We trained the model on 1 - 21 July 2022 data and tested it on 22 - 31 July for Bhopal city. When we calculate the above mentioned difference for each (hour x store) pair, we observe that there are 216 (hour x store) pairs in testing data where the xgboost model is performing better than the baseline results, and for the remaining 187 (hour x store) pairs, the baseline approach is doing better. The distribution is shown in the plot [1.4](#).

The similar plot can be observed when we calculate the difference for each (day x store) pair as shown in the figure [1.5](#).

Error During Rush Hours or Festivals

Train the xgboost model on Delhi for data 17-23 December 2021 and test it on each one hour window of 25th December to see the comparison between the model and the baseline

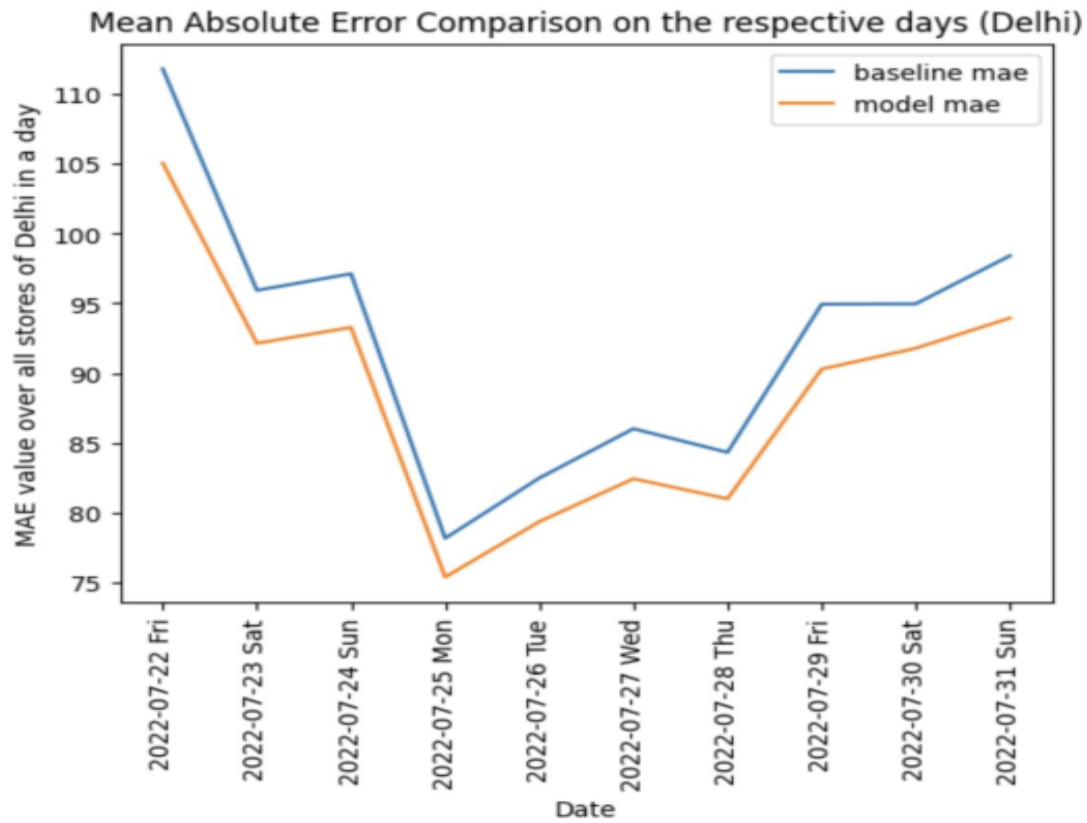


Figure 1.3: MAE Comparison Saywise

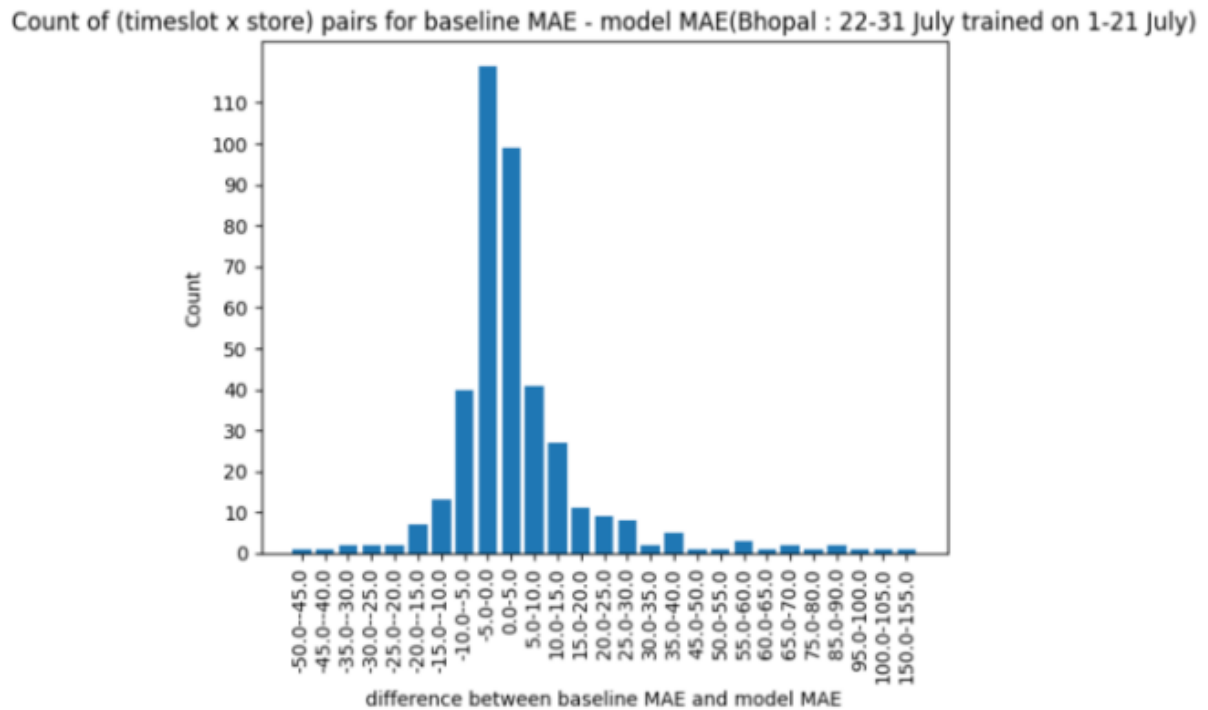


Figure 1.4: Difference of XGBoost and Baseline approach errors on (hour x store) pairs

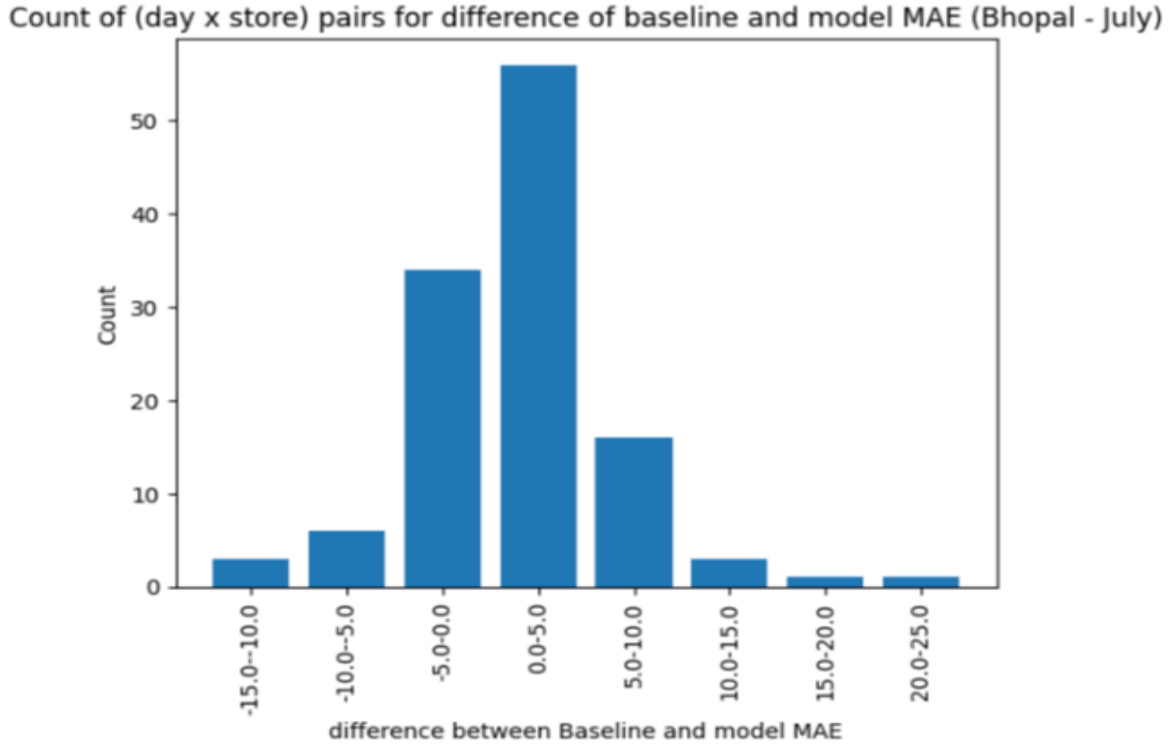


Figure 1.5: Difference of XGBoost and Baseline approach errors on (day x store) pairs

during rush hours. The comparison is shown in the figure [1.6](#). It shows that Model RMSE is lower than the baseline RMSE during rush hours, it means XGBoost model is performing better during rush hours.

Similar plot can be seen in figure [1.7](#) where the comparison is done on each hour of 24th July (Sunday) after training the model on 1-21 July.

Accuracy on boolean results

We define "x second accuracy" as the percentage of "correct predictions" with the number of total predictions, where a correct prediction is when the absolute difference between the prediction and the actual value is less than x seconds. When we calculated the "x-second accuracy" for different values of x, we observed that the accuracy in the baseline approach is better than the xgboost model for short values of x (5-10-20 seconds), but the accuracy of the xgboost model increases for large values of x (60-80 seconds) (refer figures: [1.8](#), [1.9](#), [1.10](#), [1.11](#), [1.12](#), [1.13](#)).

A Similar pattern was observed for Delhi.

model vs baseline comparison on festival day (25th Dec) in busiest restaurant of Delhi

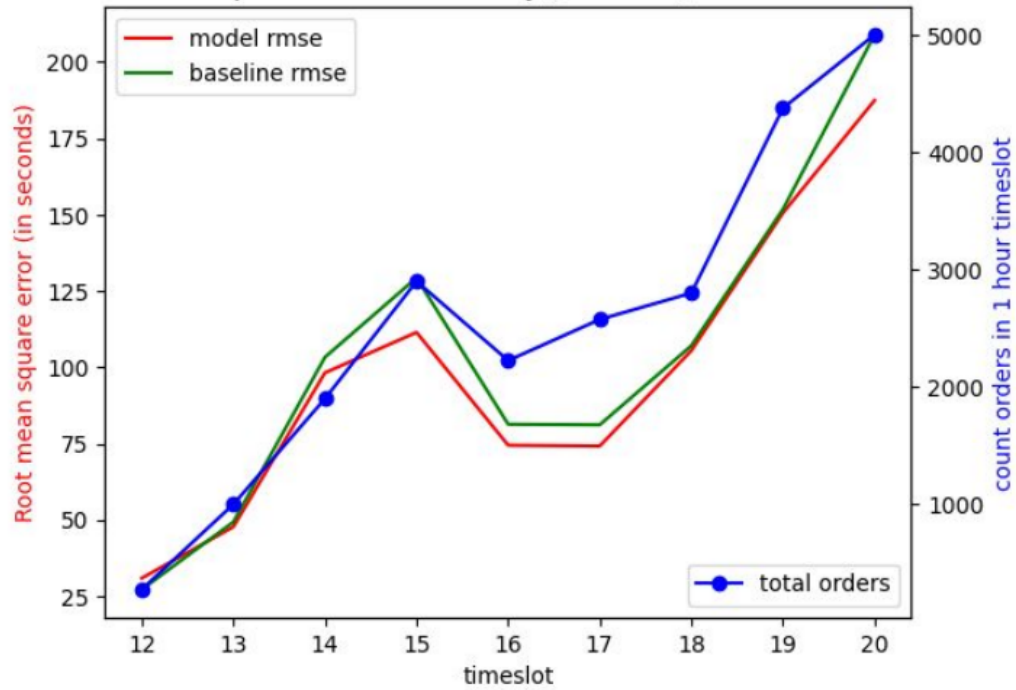


Figure 1.6: Difference of XGBoost and Baseline approach errors on (day x store) pairs

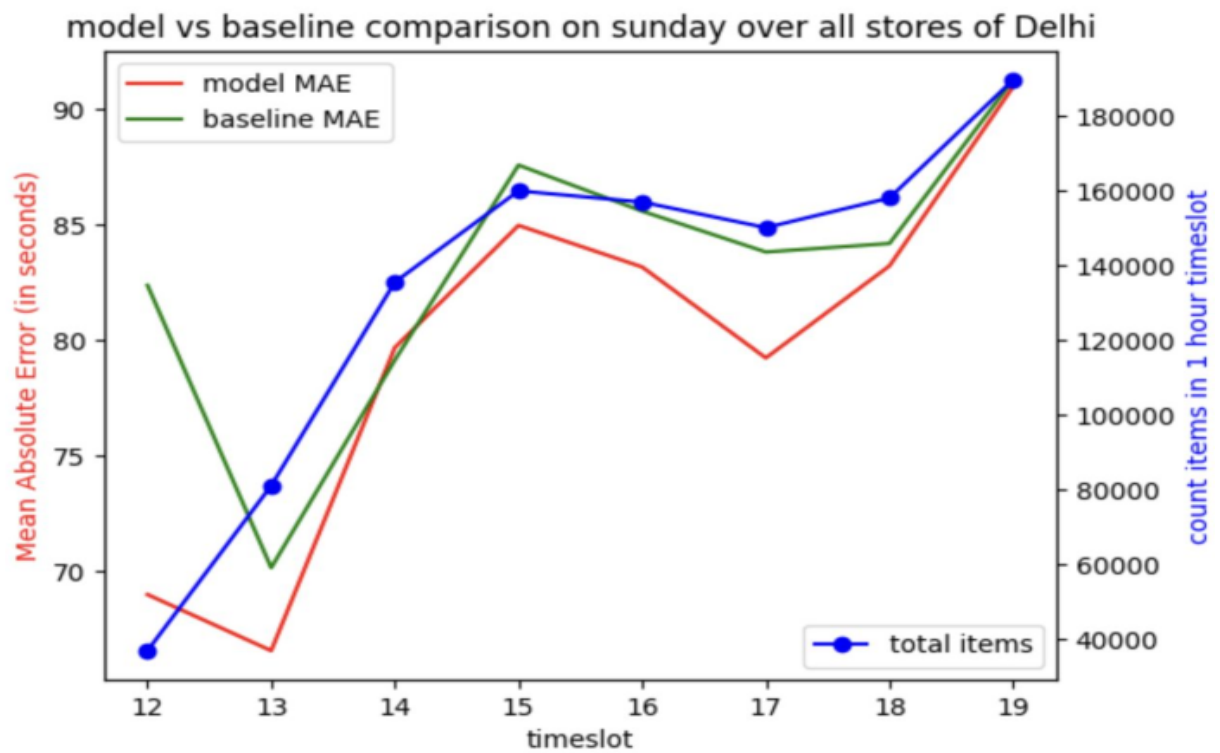


Figure 1.7: Difference of XGBoost and Baseline approach errors on (day x store) pairs

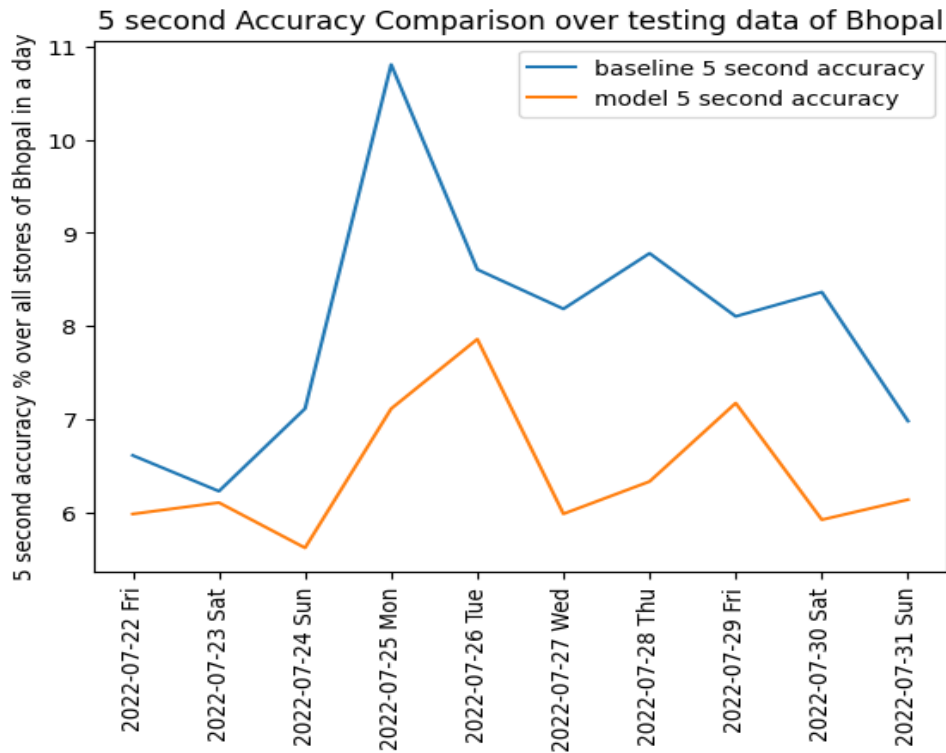


Figure 1.8: 5-second accuracy comparison between the XGBoost model and baseline approach

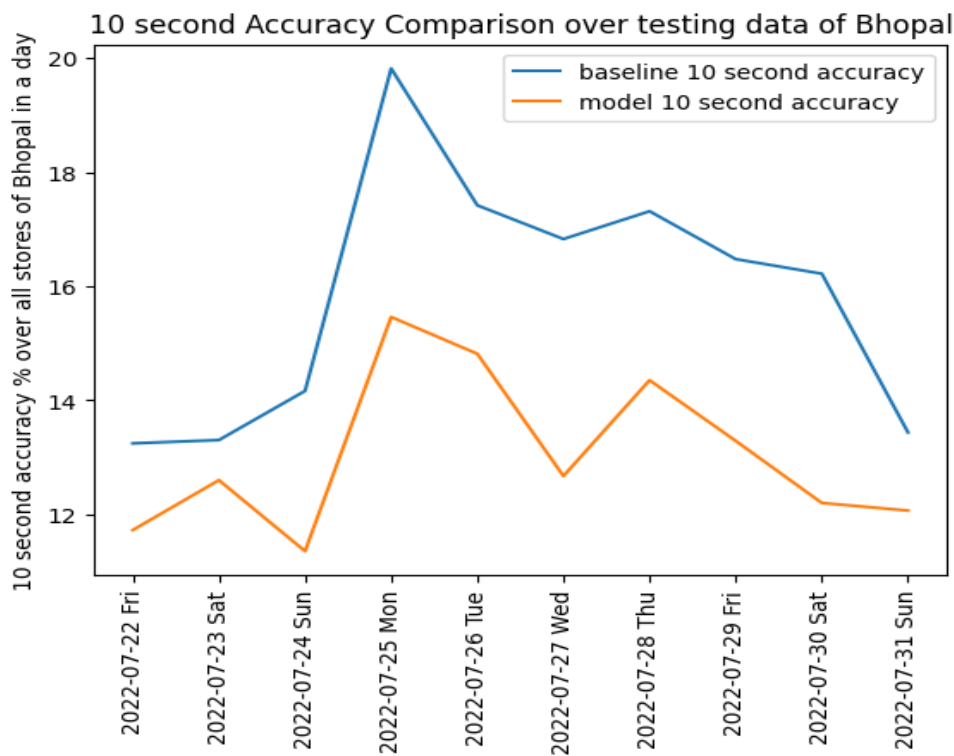


Figure 1.9: 10-second accuracy comparison between the XGBoost model and baseline approach

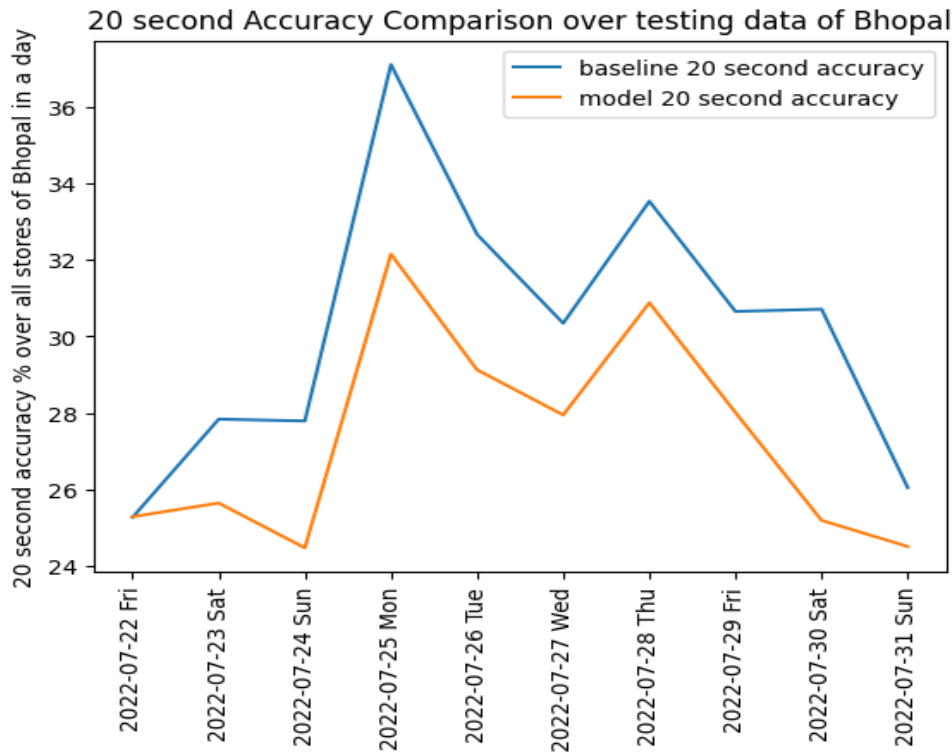


Figure 1.10: 20-second accuracy comparison between the XGBoost model and baseline approach

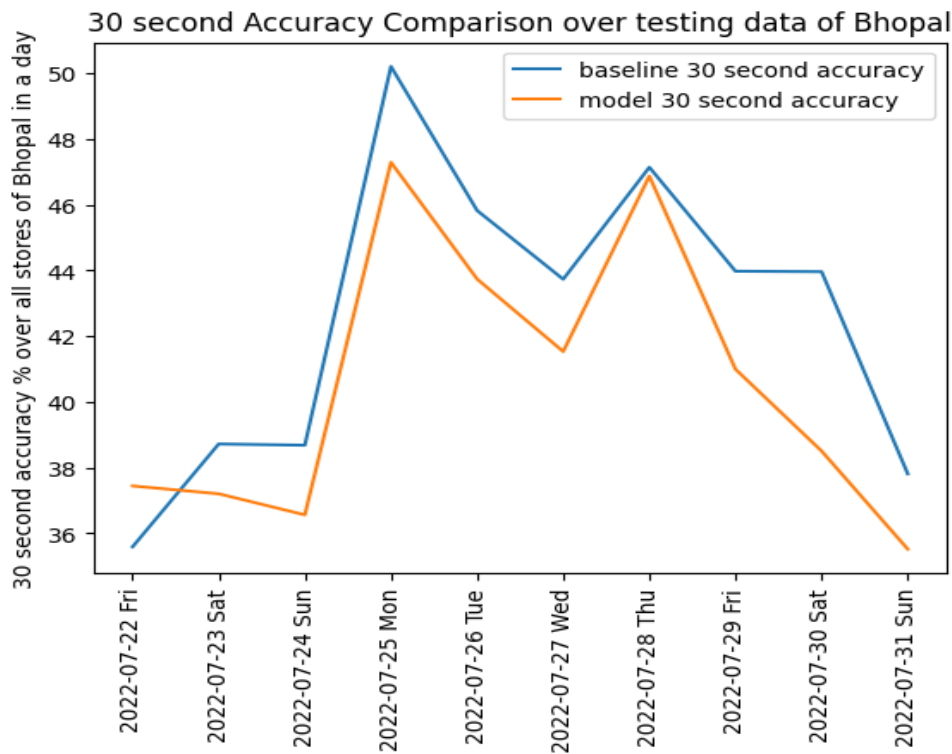


Figure 1.11: 30-second accuracy comparison between the XGBoost model and baseline approach

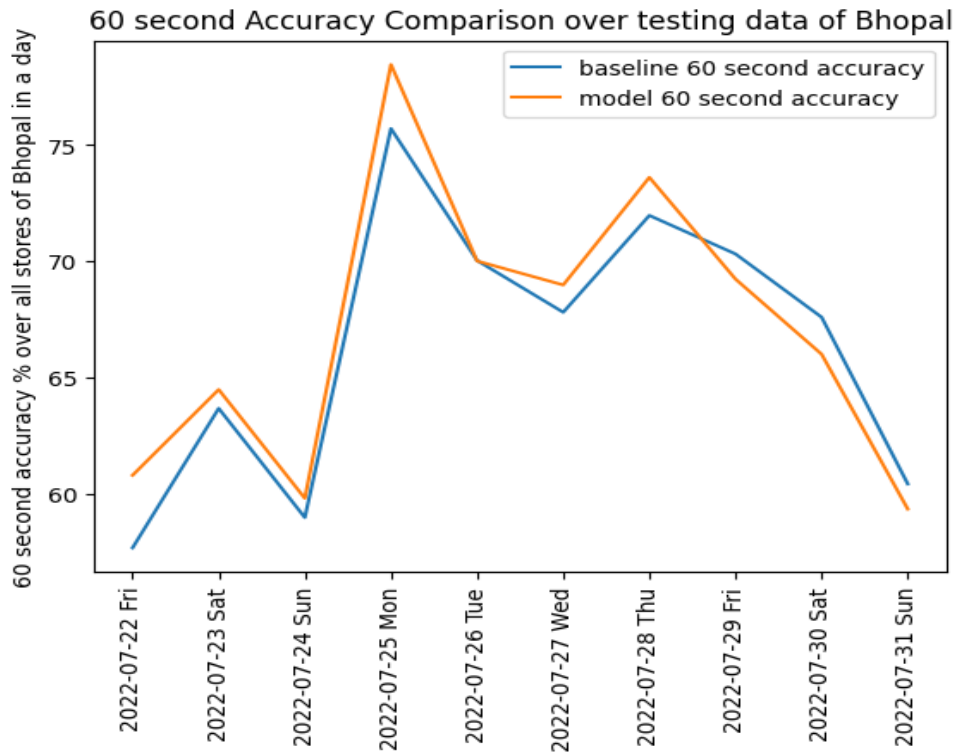


Figure 1.12: 60-second accuracy comparison between the XGBoost model and baseline approach

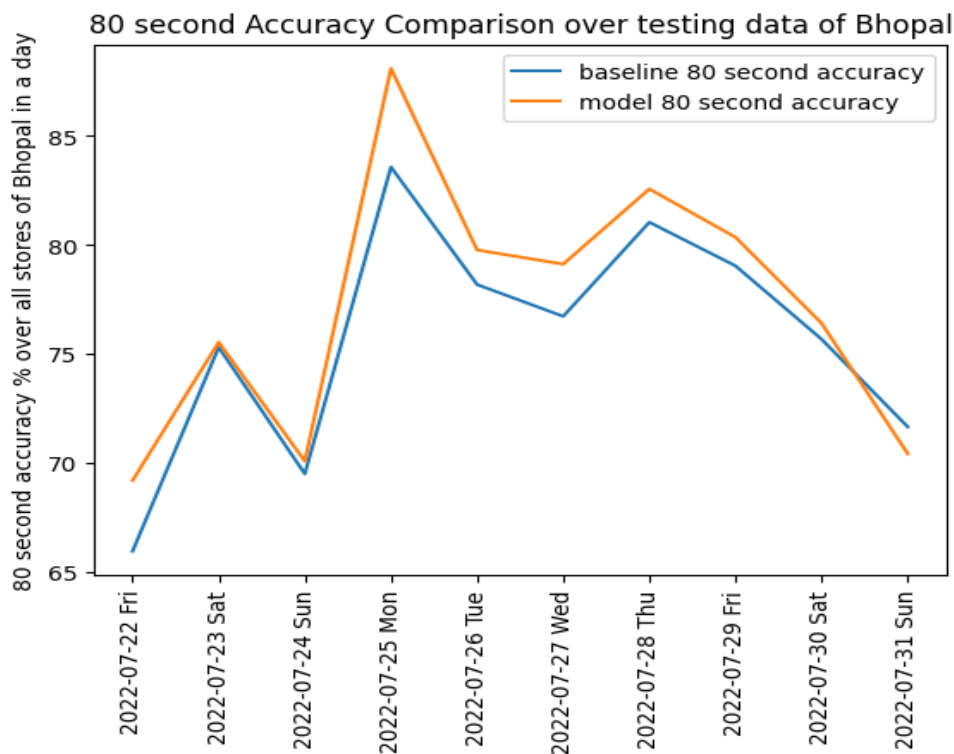


Figure 1.13: 80-second accuracy comparison between the XGBoost model and baseline approach

1.7.3 Statistical Significance of the Comparison

We define the null hypothesis as follows:

Null Hypothesis: Both models (XGBoost Model and baseline) are equally good. (The XGBoost model is not better than the baseline model).

The lower the p-value is, the lower the probability of getting that result if the null hypothesis were true. [12] A result is said to be statistically significant if it allows us to reject the null hypothesis. All other things being equal, smaller p-values are taken as stronger evidence against the null hypothesis.

We assume that for this null hypothesis, if we get $p\text{-value} < 0.005$, then it is strong evidence against the null hypothesis. Ie. if $p\text{ value} < 0.005$, then we can say that XGBoost model and baseline both are not equally good. Either one of them is better than the other.

1. We calculate the t-statistic and p-value on the predicted time of the XGBoost model and the baseline on the testing data and got the $p\text{-value} = 4.97723 \times 10^{-51}$. (See figure 1.14) Here, the p-value is significantly low, which is strong evidence against the null hypothesis. P-value denotes the probability of getting exact same prediction from both models.

```
print(stats.ttest_ind(df.predicted_prep_time.values.tolist(),df.baseline.values.tolist()))
Ttest_indResult(statistic=15.033228388881307, pvalue=4.977231414182558e-51)
```

Figure 1.14: t-statistic and p-value on the predicted time of the XGBoost model and the baseline on the testing data

2. Now, we divide the testing data into two parts:
 - (a) Normal Time: While placing an order, the following conditions are true:
 - i. Lift on the restaurant in the last 30 minutes is ≥ 0.7 and ≤ 1.3 . Here the lift is the ratio of no. of items ordered in last 30 minutes with the no. of items received in the last 60 to 30 minutes.
 - ii. the pending items in the queue ≤ 2
 - (b) Rush Time: All other times

For this, the ratio of normal time items and the rush time items is 3 : 7 in testing data.

Here we can check that the p-value is lesser during rush time. $\Rightarrow$ probability of getting null hypothesis wrong is more during rush time.

Now, check if the error in the XGBoost model is less than the error in the predictions of the baseline approach. For this, we take the following alternate Hypothesis:

Alternate Hypothesis: XGBoost Model results show less error than baseline results. (error in model prediction is less than baseline errors.)

```

: # 3. p value for items on the normal time
normalTimeItems = df[(df['lift'] >= 0.7) & (df['lift'] <= 1.3) & (df['queueLength'] <= 2)]
print(stats.ttest_ind(normalTimeItems.predicted_prep_time.values.tolist(), normalTimeItems.baseline.values.tolist()))

Ttest_indResult(statistic=9.240705617965563, pvalue=2.583314671612708e-20)

: # 3. p value for items on the rush time
rushTimeItems = df[(df['lift'] < 0.7) | (df['lift'] > 1.3) | (df['queueLength'] > 2)]
print(stats.ttest_ind(rushTimeItems.predicted_prep_time.values.tolist(), rushTimeItems.baseline.values.tolist()))

Ttest_indResult(statistic=12.587795322457385, pvalue=2.6692178593928908e-36)

```

Figure 1.15: t-statistic and p-value on the predicted time of the XGBoost model and the baseline on testing data during Normal Time and Rush Hours

We calculate the t-statistic and p-value on the absolute error in the XGBoost model prediction and the absolute error in the baseline prediction with the actual KPT:

1. On whole testing data (22-31 July 2022): We get the p-value = 0.0007 (figure : [1.16](#)). The p-value denotes the probability of getting our alternate hypothesis wrong. Since, p-value < 0.005, this is a strong evidence that the deviation from the actual KPT is less in the XGBoost model as compared to the baseline approach results.

```

: # 1. p value on the whole test data for alternative hypothesis that the mean
# of error is significantly less than the baseline error
print(stats.ttest_ind(df.abs_error_model.values.tolist(), \
                      df.abs_error_baseline.values.tolist(), \
                      alternative='less'))
# low p-value indicates strong evidence against the null
# hypothesis and suggests that the mean of the absolute model error is significantly
# less than the mean of the absolute baseline error.

Ttest_indResult(statistic=-3.1920756186928747, pvalue=0.000706462592757283)

```

Figure 1.16: t-statistic and p-value for the alternate hypothesis on whole testing data

2. During Normal Times: We get p-value = 0.2124. Here, the p-value is much higher than the other cases. Since the p-value > 0.005, we can not firmly say that the XGBoost model will show less error deviation from the actual KPT. (figure: [1.17](#))

```

: # 2. p value on items during normal times for alternative hypothesis that the mean
# of error is significantly less than the baseline error
print(stats.ttest_ind(normalTimeItems.abs_error_model.values.tolist(), \
                      normalTimeItems.abs_error_baseline.values.tolist(), \
                      alternative='less'))
# low p-value indicates strong evidence against the null
# hypothesis and suggests that the mean of the absolute model error is significantly
# less than the mean of the absolute baseline error.

Ttest_indResult(statistic=-0.7979840541750993, pvalue=0.2124425929975221)

```

Figure 1.17: t-statistic and p-value for the alternate hypothesis during normal times

3. During Rush Times: We get p-value = 0.0006. Here the p-value is very low during rush times. Here p-value < 0.005 hence, this is strong evidence of getting our model better during rush times. (Figure : [1.18](#))

1.7.4 Model Deterioration Window

there is no significant deterioration. We could only see marginal deterioration since we had only one month of continuous data.

```

: # 2. p value on items during rush times for alternative hypothesis that the mean
# of error is significantly less than the baseline error
print(stats.ttest_ind(rushTimeItems.abs_error_model.values.tolist(), \
                      rushTimeItems.abs_error_baseline.values.tolist(), \
                      alternative='less'))
# low p-value indicates strong evidence against the null
# hypothesis and suggests that the mean of the absolute model error is significantly
# less than the mean of the absolute baseline error.

Ttest_indResult(statistic=-3.2089935700675505, pvalue=0.0006662661659922975)

```

Figure 1.18: t-statistic and p-value for the alternate hypothesis during rush times

We can also check what is the rate of deterioration approximately by training the model on 1-21 July and then testing it on 28 July. Here the difference between the training data and the testing data is 7 days. Later, we train the model for 7-27 July and test it on 28 July. Here the difference between the training and testing data is 1 day only. All other parameters in the training are the same. Hence, we can see the effect of deterioration by comparing both results.

Table 1.5: Error values (in seconds) on 28 July 2022 Bhopal data for different training periods for model deterioration check

Error Evaluation Type	Training Period	
	1 - 21 July 2022	7 - 27 July 2022
RMSE	71.87645082751689	71.44089240996973
MAE	48.92031708364018	48.61247639176124
Std Dev	52.65953626833944	52.35100999401375
Median Error	32.40373510742187	32.05204382324219

Table 1.6: Error values (in seconds) on 28 July 2022 Bhopal data for different training periods for model deterioration check

Error Evaluation Type	Training Period	
	1 - 21 July 2022	7 - 27 July 2022
RMSE	110.67000087856664	110.19615091293166
MAE	81.13496527079111	80.9877360907821
Std Dev	75.26597175994914	74.72735964100058
Median Error	63.02193750000001	62.804522491455074

Table 1.5 and 1.6 shows that there is a very slight deterioration in both Bhopal and Jaipur cities. In the first case, we trained the model from 1-21 July and calculated the errors for 28th July (difference of 7 days), whereas when we train the model for 7-27th July and calculate the errors for 28th July (difference of 1 day), we got a slight reduction in the errors.

Chapter 2

Service Implementation for JFL

2.1 Introduction

A trained machine learning model should be deployed in such a way that it can handle lots of requests without any downtime. The remainder of this thesis will delve into the deployment of the machine learning model and the implementation of the server infrastructure.

While implementing a service infrastructure for this, several points should be taken care of. Here are some key considerations:

1. **Scalability:** The service infrastructure should be designed to handle a high volume of incoming requests efficiently. As the number of users and prediction requests increases, the system should scale seamlessly to meet the demand without sacrificing performance.
2. **Real-time Prediction:** The infrastructure should support real-time predictions, allowing users to obtain accurate estimates quickly. The system should be able to process and respond to prediction requests promptly, ensuring minimal delays in providing the estimated preparation time.
3. **API Design:** The APIs exposed by the service infrastructure should be well-designed and intuitive. They should provide a clear and consistent interface for users to submit prediction requests and receive the corresponding results. Proper documentation and versioning of APIs should be maintained to support easy integration and future enhancements.
4. **Data Integration:** The infrastructure should facilitate seamless integration with the data sources required for prediction. This may include connections to the store's database or data streams that provide the live order informations and information on all relevant factors influencing preparation time. Data integration should be robust, reliable, and capable of handling real-time updates.
5. **Performance and Reliability:** The service infrastructure should be designed to deliver high-performance and reliable predictions. This requires optimizing the system for low latency and high throughput, as well as implementing mechanisms to handle failures gracefully, such as automatic error handling and fault tolerance.

2.2 Current Architecture of JFL & Requirements

The Dominos India Team (JFL Product Team) currently, JFL have two main servers:

1. Event Publisher: It publishes real-time events for all orders whenever an order state changes. It contains the following information for an order:
 - (a) Order State (order created, order prepared, order delivered, etc)
 - (b) Information of each item in an order (item type, item name, bill etc)
 - (c) Timestamps (order placed time, kitchen display time, oven time (if order prepared or delivered), and customer details.
2. JFL-OAA (JFL - Order Allocation Algorithm): It allocates the order to a nearby store to the customer.

2.3 Requirements of Dominos Team

Since adding a machine learning model directly to the JFL-OAA might be a risky task as it needs to be tested on the real-time production data. For this, the JFL Team puts the following requirements to the IIT-Delhi team:

1. JFL Team wants to show the predicted time to the customer even before creating the cart (adding items to the cart).
2. Real-time queries to machine learning model each time might be slow, hence they want to pre-predict the preparation time of the each types of orders for each store and store them in cache.

We define the types of orders in Dominos as follows:

- (a) Generic Order (less than or equal to 3 baked items)
 - (b) Bulk Order (more than 3 baked items)
3. They do not want to increase the load on their own database initially, hence to not increase the requests per minute on the main JFL database, they want us to listen their order events from Event publisher and consume it on our own.
4. They want an interface where they can hit the API to get the predicted preparation time of a store for both types of orders. Also, the service should use Java springboot framework since all of their source code is written in Java.

To fit our model in their current architecture, we created a micro-service architecture in java using Springboot framework and expose some APIs for JFL-OAA to call.

2.4 New Architecture Design: Microservices or Monolithic?

Microservices architecture is followed to implement the service. It is because Microservice architecture offers several benefits over traditional monolithic server architectures. Here are some key advantages [3] :

1. **Scalability:** Microservices allow for independent scaling of individual services. In a monolithic architecture, the entire application needs to be scaled even if only a specific component requires more resources. With microservices, we can allocate resources where they are needed, leading to better resource utilization and cost efficiency.
2. **Flexibility and Agility:** Microservices enable flexibility in development and deployment. Each service can be developed, deployed, and scaled independently, which allows for faster development cycles and the ability to adopt new technologies or languages for specific services without impacting the entire application. This flexibility enhances agility, making it easier to iterate and introduce new features or improvements.
3. **Fault Isolation:** In a monolithic architecture, if a single component fails, it can potentially bring down the entire application. Microservices provide fault isolation since each service operates independently. If one service fails, it does not affect the functioning of other services, leading to higher availability and fault tolerance.
4. **Independent Development and Deployment:** Microservices enable different teams to work on different services independently. This promotes parallel development and allows teams to focus on specific functionalities, resulting in faster development cycles and better organizational scalability. Additionally, each service can be deployed independently, reducing the risk of deployment failures and enabling continuous delivery practices.
5. **Technology Heterogeneity:** Microservices allow for the use of different technologies, frameworks, and programming languages for each service. This flexibility enables teams to choose the most suitable tools for their specific service requirements, making it easier to leverage the strengths of different technologies and libraries. Now, we can use python for the model to train and predict, and java springboot for API implementations.
6. **Ease of Maintenance and Evolvability:** Microservices simplify maintenance and evolution of applications. With smaller, focused services, it becomes easier to understand, test, and modify specific parts of the system. Additionally, as services are decoupled, making changes to one service does not require changes in others, reducing the risk of unintended consequences.
7. **Fault Tolerance and Resilience:** Microservices architecture promotes the use of techniques like redundancy, replication, and failover mechanisms. This helps in building resilient systems that can handle failures and provide high availability. Services can be distributed across different servers or even geographical locations, reducing the impact of hardware or network failures.

These are some advantages of implementing a micro-service architecture instead of a monolithic architecture.

Apart from this, we used "Singleton Design Pattern" in the services. It is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance.

2.5 System Architecture Design

We divide the whole system to the following 3 microservices:

1. KPT Model (The ML Model)
2. KPT Handler (Handler)
3. Get Prediction (Interface for JFL)

The working and the dependency of each other on the other is explained in the figure

2.1

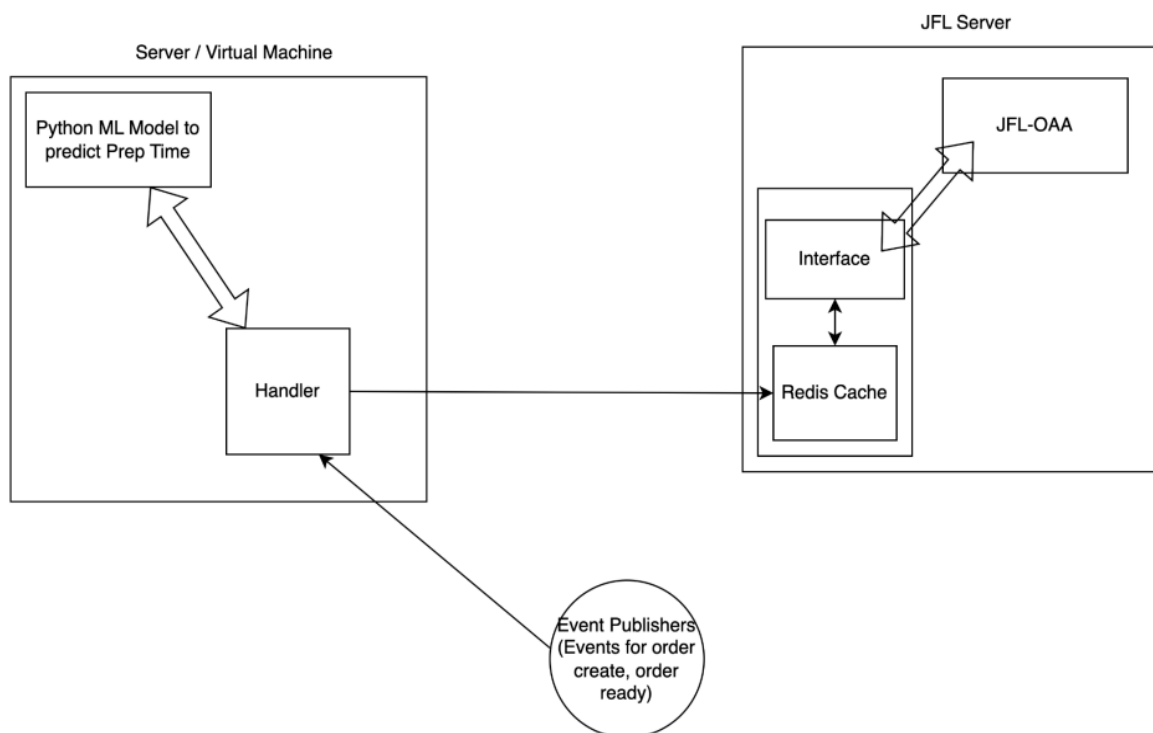


Figure 2.1: Architecture Design of Microservices

KPT Handler and KPT Model can communicate to each other whereas KPT Handler can also consume the events from the event publisher. It writes the predictions to the Redis cache which are read by the Interface when JFL-OAA makes a request to Interface (GetPrediction).

2.5.1 KPT Model

This microservice loads the trained machine learning model and gives the predictions of requests. This microservice is implemented in Python [11] using Flask framework. It has one exposed API, which takes the values of all features of the model.

It returns two prediction values (in a json array). First value contains the kitchen preparation time if order type is a generic order. Second value is the kitchen preparation time for a bulk order in the same store at that time.

Endpoints

localhost:5040/prediction HTTP GET or POST requests

Path Variables

1. f1 : store Id
2. f2 : day of week (Integer)
3. f3 : timeslot
4. f4 : count past orders in last 30 minutes in same store
5. f5 : Lift in Orders (ratio of past orders in 30 minutes with last 60-30 minute window)
6. f6 : Avg prev KPT in last 60 minutes
7. f7 : Avg Prev KPT in last 7 days
8. f8 : Avg KPT in the same timeslot in the same day in last 1 month.
9. f9 : length of the queue (unbaked orders) in the store

Sample curl command to hit the endpoint:

```
curl --location \
'localhost:5040/prediction?f1=DPI67072&f2=5&f3=15&
f4=18&f5=1.0&f6=601.0&f7=601.0&f8=601.0&f9=6'
```

Sample Response:

```
[
  105.61868286132812,
  169.25038146972656
]
```

Note: This API is called by KPT Handler microservice which is described next.

2.5.2 KPTHandler

KPTHandler is a microservice implemented in Java with Springboot framework [8]. It has internal mysql database to store the orders information. which does the following:

1. It listens the events of JFL Event listener to consume order events. In order to be consumed the order events, JFL Event publisher should hit POST requests to the endpoint `postEvent` on port 8083.

If event type is :

- (a) Order created: Adds new entries for each item of the order to the database, changes the store state by updating the load on the store and other factors. After this, make a prediction call to "KPT Model" service and update the newly received prediction times to redis cache of "Get Prediction" (Interface) service.

- (b) Order Prepared: Updates the entries of each item of the order to the database with their oven time. Then, it changes the store state by updating the load (decreasing) and all other features.

Now, make a prediction call to "KPT Model" service and update the newly received prediction times to redis cache of "Get Prediction" service.

The flow of KPT Handler can be best described via the flowchart 2.2

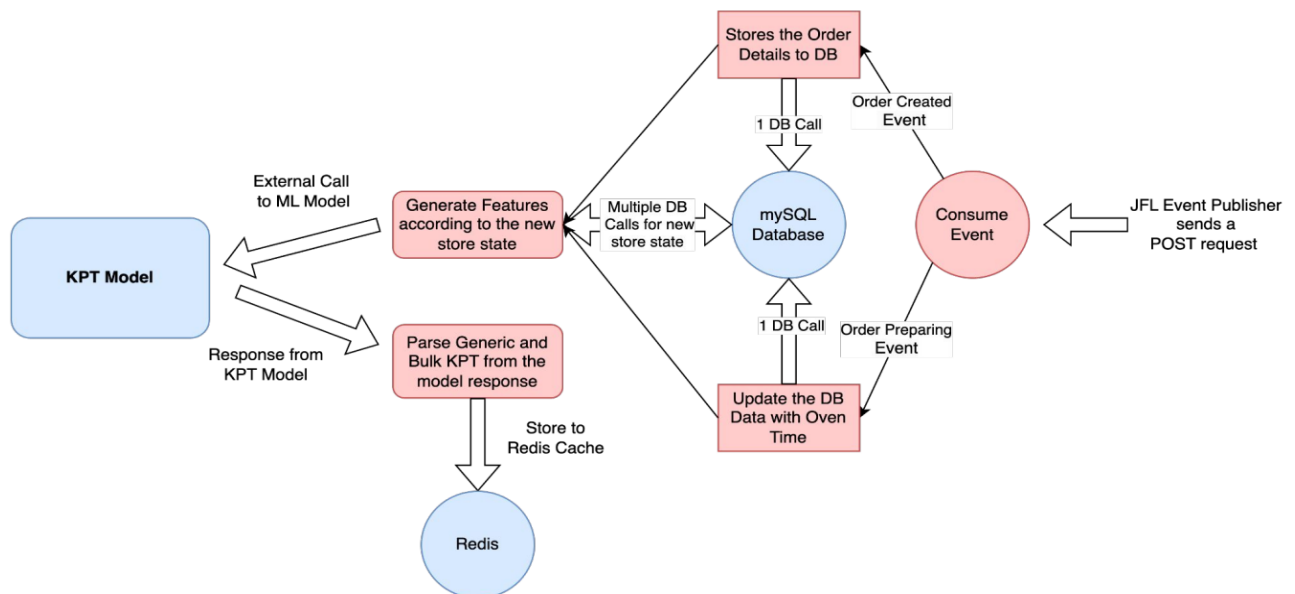


Figure 2.2: Architecture Design of KPT Handler Microservice

Endpoints

To publish an event to KPT Handler, JFL Event publisher needs to hit the `postEvent` API of KPT Handler with their JSON data of the order event via HTTP POST request. It consumes the event and returns nil or error (in case of failure).

Sample curl command to hit the endpoint:

```
curl --location 'http://localhost:8083/postEvent' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
    "LocationCode": "DPI67072",  
    "OrderDate": "2020-10-16",  
    "EventType": "ORDER_CREATED",  
    "Timestamp": "2020-10-16 14:49:20",  
    "Data": {  
        "Order": {  
            "Orders": [  
                {  
                    "Location_Code": "DPI67072",  
                    "Order_Number": 51,  
                    "Order_Date": "2020-10-16T00:00:00",  
                    "Customer_Name": "XXXX",  
                    "Actual_Order_Date": "2020-10-16T14:49:20.487",  
                    "Order_Saved": "2020-10-16T14:49:20.487",  
                    "Return_Time": null,  
                    "Delivery_Time": null,  
                    "Kitchen_Display_Time": "2020-10-16T14:49:20.487",  
                    "Total_Amount": 450.0000  
                }  
            ],  
            "Order_Lines": [  
                {  
                    "Location_Code": "DPI67072",  
                    "Order_Number": 51,  
                    "Order_Date": "2020-10-16T00:00:00",  
                    "Line_Number": 1,  
                    "Order_Line_Status_Code": 1,  
                    "Oven_Time": null,  
                    "Quantity": 1.0,  
                    "Menu_Code": "PIZ0128",  
                    "Size_Code": "BHT95",  
                    "Orig_Menu_Code": "PIZ0118",  
                    "Base_Price": 305.0000,  
                }  
            ],  
        },  
    },  
},
```

```
}'
```

This is a sample order event of or type order created. It contains whole information about the order, but in this query, we have not added the whole information as we do not need the remaining fields.

If the KPT Handler service consumes it successfully, then it returns a string "order created". See figure [2.3](#)

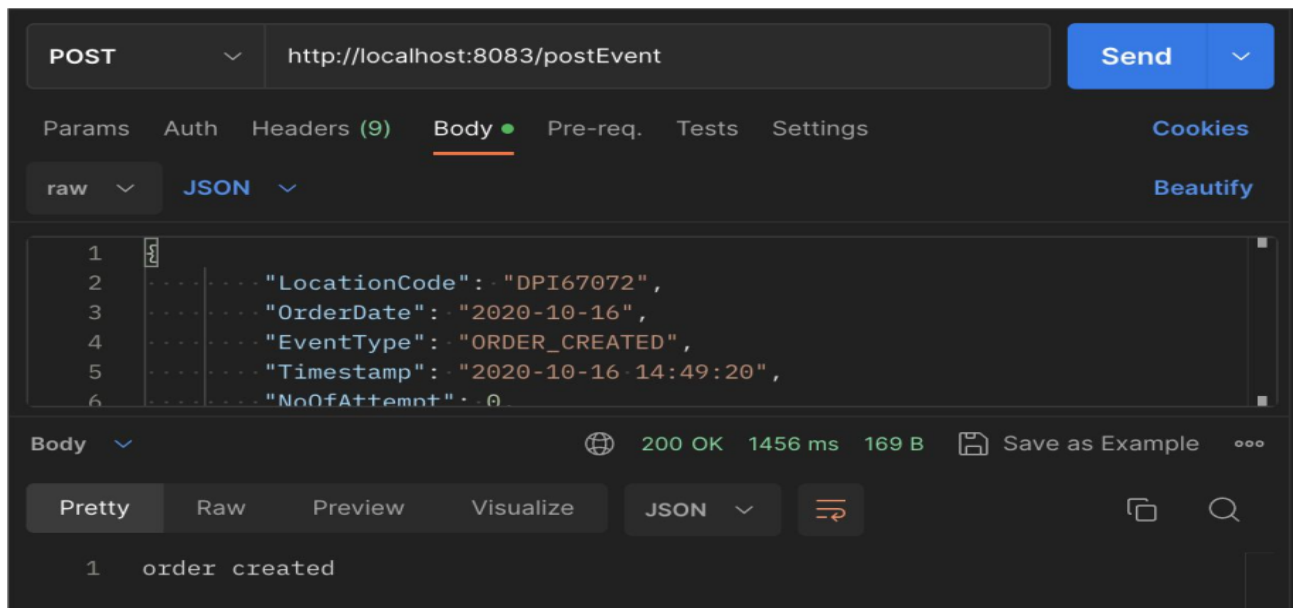


Figure 2.3: Event Publisher hitting KPT Handler

2.5.3 Get Prediction (Interface)

This microservice contains one endpoint which is being hit by JFL-OAA to get the predicted kitchen preparation time of a store for generic order and bulk orders. This service contains Redis cache which is being filled by the KPT Handler micro-service.

Endpoints

As mentioned above, this interface is only used to read the Redis cache. Hence, it has only one endpoint "getPrediction" which reads generic and bulk order predicted KPT for the store. Store id is passed with as follows:

```
localhost:8080/getPrediction/{storeId}
```

Sample curl command to hit the endpoint:

```
curl --location 'localhost:8080/getPrediction/DPI67072'
```

Sample Response:

```
[  
  105.61868286132812,  
  169.25038146972656  
]
```

Here first value is predicted kitchen preparation time for a generic order and second value is for bulk order in store with store id "DPI67072".

2.6 Docker for service Deployments

Docker allows us to automate the deployment inside lightweight, isolated containers. It provides a consistent and reproducible environment for running software, making it easier to package and distribute services with their dependencies. [1]

In traditional server deployments, applications are typically run directly on the host operating system. However, this approach can lead to issues with compatibility between different software dependencies and configurations, making it challenging to ensure consistent behavior across different environments. This is where Docker comes in. Docker containers are highly portable. Once the application is packaged into a container, it can be deployed on any system that supports Docker, whether it's a developer's machine, or a production environment. This portability streamlines the deployment process and makes it easier to move applications between different environments.

Appendix A

XGBoost Model for Regression

XGBoost, short for eXtreme Gradient Boosting, is a powerful machine learning algorithm known for its high performance and scalability in both classification and regression tasks. It is based on the gradient boosting framework and has gained popularity due to its ability to handle diverse data types, feature interactions, and missing values effectively.

At its core, XGBoost builds an ensemble of weak prediction models, typically decision trees, to create a strong predictive model. The algorithm iteratively trains a sequence of trees where each subsequent tree aims to correct the mistakes made by the previous trees. This boosting process enables the model to learn complex relationships and make accurate predictions [4].

Here are the key steps involved in the XGBoost algorithm for regression:

1. **Data Representation:** XGBoost uses a custom data structure called a "DMatrix" to store the training data. It efficiently handles both dense and sparse datasets and supports additional functionalities like missing values and weightings.
2. **Objective Function:** XGBoost uses a specific objective function for regression tasks, typically the mean squared error (MSE) or the mean absolute error (MAE). The objective function measures the difference between the predicted and actual values and guides the learning process.
3. **Tree Ensemble Construction:** XGBoost builds decision trees in a greedy manner, optimizing the objective function at each step. It starts with a single tree and adds additional trees iteratively. Each new tree focuses on the residuals (the differences between the predicted and actual values) of the previous trees, aiming to reduce these residuals further.
4. **Regularization:** XGBoost includes regularization techniques to control the complexity of the model and prevent overfitting. It uses both L1 (Lasso) and L2 (Ridge) regularization terms in its objective function to penalize large coefficients and encourage feature selection.
5. **Tree Construction:** XGBoost constructs decision trees by recursively partitioning the data based on the features. It uses a technique called "gradient boosting" to determine the optimal splits by considering the gradients of the loss function. It intelligently explores different split points and depths to capture complex patterns and avoid overfitting.
6. **Tree Pruning:** To further improve generalization, XGBoost applies tree pruning techniques to remove unnecessary branches and reduce the model complexity. Pruning helps prevent overfitting and promotes better generalization performance on unseen data.

7. **Ensemble Prediction:** Once all the trees are constructed, XGBoost combines their predictions to produce the final output. The model assigns a weight to each tree's prediction, considering its performance during the training process. The weighted sum of these predictions forms the overall prediction of the XGBoost model.
8. **Regularization Parameters:** XGBoost provides several hyperparameters that control the learning process, such as the learning rate, maximum tree depth, subsampling ratio, and column subsampling ratio. These parameters allow you to fine-tune the model's behavior and balance between accuracy and complexity.

XGBoost has proven to be highly effective in various regression tasks, demonstrating superior performance in both predictive accuracy and computational efficiency [2]. It has become a popular choice among data scientists and machine learning practitioners for solving regression problems.

Appendix B

Springboot Framework for Java

Java Spring Boot is a framework designed to simplify the development of Java applications, particularly web-based and microservices-based applications. It builds upon the Spring framework, providing a streamlined approach to application development. Spring Boot offers a range of features and benefits that make it a popular choice for developers. It facilitates rapid application development by providing auto-configurations and starter dependencies that handle much of the configuration and integration work.

It supports the development of microservices architectures, enabling easy communication between services. Additionally, Spring Boot includes an embedded server, eliminating the need for external server deployment. It offers dependency management through starter dependencies and integrates seamlessly with the broader Spring ecosystem, allowing developers to leverage other Spring projects. Spring Boot Actuator further enhances the framework by providing monitoring and management capabilities for applications. Overall, Spring Boot simplifies Java application development, allowing developers to focus on business logic rather than infrastructure concerns [8].

Bibliography

- [1] **Bashari Rad, B., H. Bhatti, and M. Ahmadi** (2017). An introduction to docker and analysis of its performance. *IJCSNS International Journal of Computer Science and Network Security*, **173**, 8.
- [2] **Bentéjac, C., A. Csörgő, and G. Martínez-Muñoz** (2019). A comparative analysis of xgboost.
- [3] **Blinowski, G., A. Ojdowska, and A. Przybyłek** (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, **10**, 20357–20374.
- [4] **Chen, T. and C. Guestrin**, Xgboost: A scalable tree boosting system. *In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. Association for Computing Machinery, New York, NY, USA, 2016. ISBN 9781450342322. URL <https://doi.org/10.1145/2939672.2939785>.
- [5] **Eliran Sherzer, o. B. D. K., Arik Senderovich** (2022). Can machines solve general queueing systems? *arXiv:2202.01729v1*.
- [6] **Elisheva Chocron, P. F., Izack Cohen** (2022). Delay prediction for managing multiclass service systems: An investigation of queueing theory and machine learning approaches. *IEEE*.
- [7] **Hirschberg, C., A. Rajko, and M. W. Thomas Schumacher** (2016). The changing market for food delivery. URL <https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-changing-market-for-food-delivery>.
- [8] **Mythily, M., A. Samson Arun Raj, and I. Thanakumar Joseph**, An analysis of the significance of spring boot in the market. *In 2022 International Conference on Inventive Computation Technologies (ICICT)*. 2022.
- [9] **Ole-Edvard Ørebæk, M. G.** (2021). Exploring the hyperparameters of xgboost through 3d visualizations. *CEUR - WS*.
- [10] **Parmar, U.** (2023). Data preparation and preparation time prediction.
- [11] **van Rossum, G. et al.** (1991–). The Python programming language. URL <http://www.python.org/>.
- [12] **Wasserstein, R. L. and N. A. Lazar** (2016). The asa statement on p-values: Context, process, and purpose. *The American Statistician*, **70**(2), 129–133. URL <https://doi.org/10.1080/00031305.2016.1154108>.
- [13] **Wikipedia** (1995 - present). Jubilant foodworks limited. URL https://en.wikipedia.org/wiki/Jubilant_FoodWorks.